

Jin-Soo Kim  
[\(jinsoo.kim@snu.ac.kr\)](mailto:jinsoo.kim@snu.ac.kr)

Systems Software &  
Architecture Lab.

Seoul National University

Jan. 6 – 17, 2020

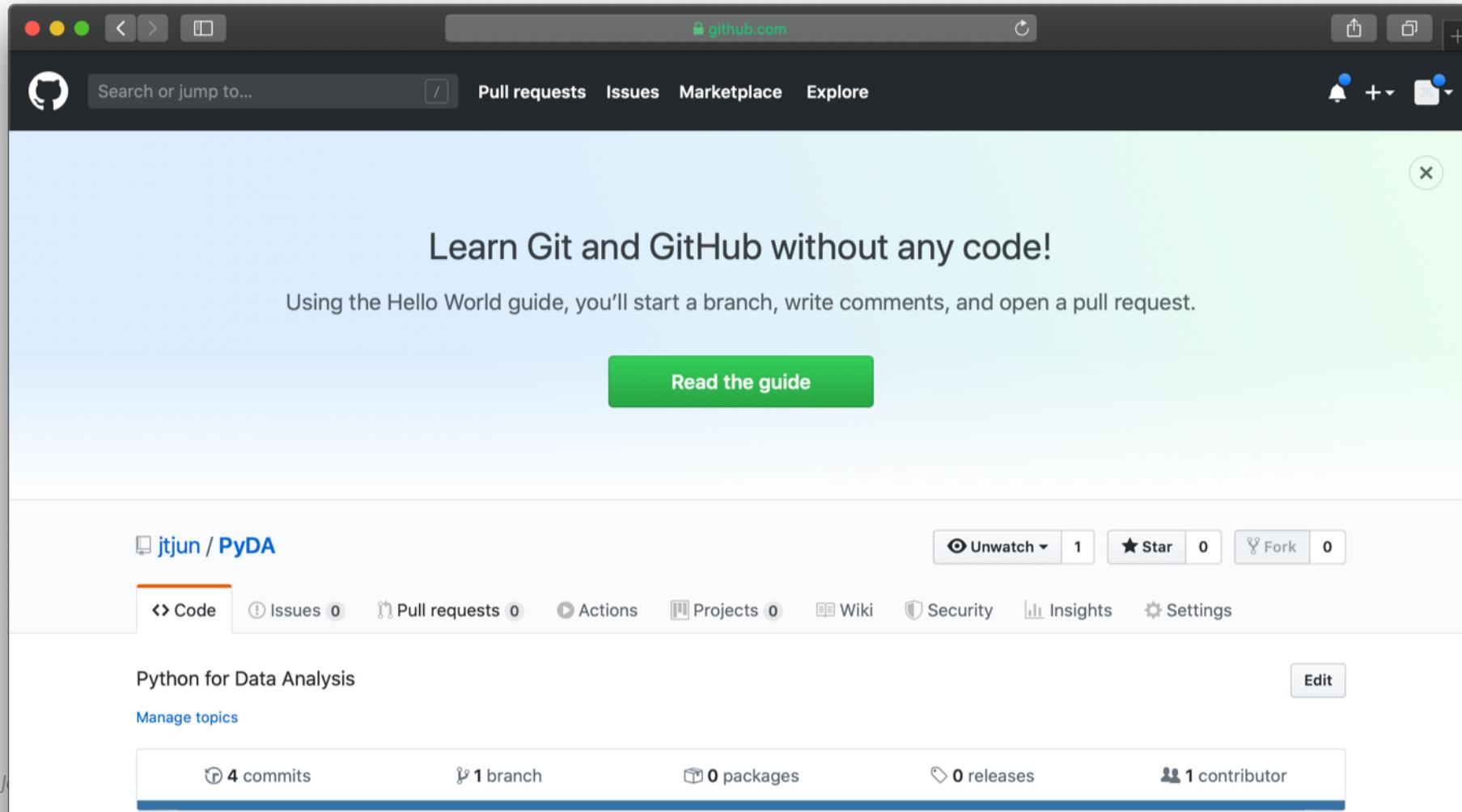


*Python for Data Analytics*

# Basic 02

# 다운로드 안내

- <https://github.com/jtjun/PyDA/archive/master.zip>

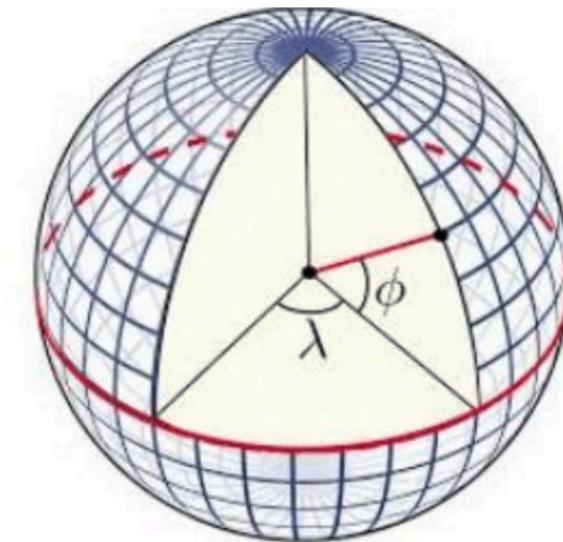
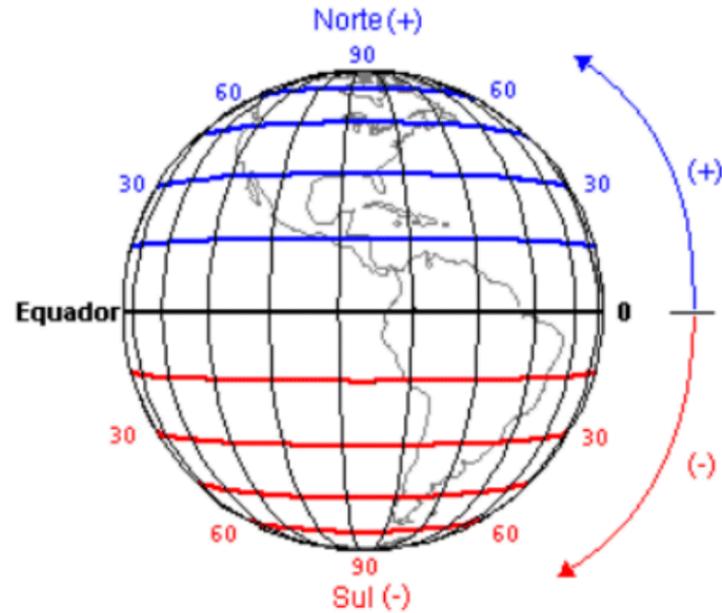
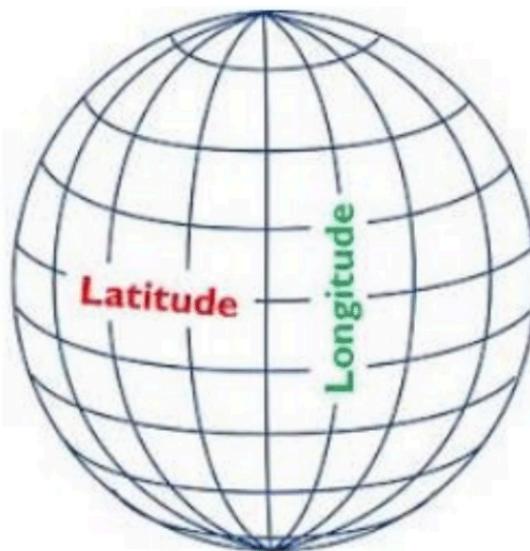


# Cities

# Lab 2-0. Cities

## ■ 지구 상에서 두 지점간 거리?

- 지구는 둥그니까~
- 직선 거리가 직선 거리가 아니다.
- 구 위의 임의의 두 점의 거리는?



# Lab 2-0. Cities

- 지구 상에서 두 지점간 거리?

- 코드로 표현하는 방법?
- math 라이브러리를 사용!
- import math → math.pi, math.sin, math.cos, math.acos

$(lat1, lon1), (lat2, lon2)$

$$\begin{aligned} \cos \theta &= \sin(lat1)\sin(lat2) + \cos(lat1)\cos(lat2)\cos(lon2 - lon1) \\ \theta &= \text{acos}(\cos(\theta)) \end{aligned}$$

$$distance = r\theta, \quad 2\pi \text{ radian} = 360^\circ$$

Radian 각도로 표현된 두 지점 사이 거리

# Lab 2-0. Cities

- 사용자에게 두 위치를 받아서 거리를 출력하기

```
def radian(d)

def dist_between_loc(loc1, loc2)

def loc_dist()
```

# Lab 2-0. Cities

- 각도를 radian 형태로 변환하기

- Parameter - d :

- type : float
    - 변환하고자 하는 각도의  $360^\circ$  형태

- Return - radian:

- type : float
    - radian 형태로 변환된 각도
    - Hint.  $360^\circ = 2\pi$

```
def radian(d)
```

# Lab 2-0. Cities

## ■ 두 지점 간의 거리 구하기

- Parameter – loc1, loc2 :
  - type : tuple (float, float)
  - latitude, longitude 가 순서대로 담긴 tuple \*immutable
- Return - dist:
  - type : float
  - 앞의 공식을 이용해 계산한 거리
  - 단, lon2 – lon1 은 degree form에서 계산 (radian 변환 전에 계산 후 변환)

```
def dist_between_loc(loc1, loc2)
```

# Lab 2-0. Cities

## ■ 사용자의 입력 처리하기

- Parameter - None :
  - 단, 사용자에게 input 을 받음 (Word Count 와 같이 input() 함수의 인자는 없음)
  - 띄어쓰기로 구분된 'lat lon'을 두 번 입력 받음
- Return - None:
  - 값을 return 하는 것이 아닌, 사용자에게 보여줌
  - 다른 것 없이 거리만 출력
- 18 page의 입출력 예시 참고

```
def loc_dist()
```

# Lab 2-0. Cities

- 여기에서 집까지 거리 구하기

```
$ python distEarth.py  
37.455813 126.954699 ↵ 첫 번째 위도 경도  
37.266653 126.999386 ↵ 두 번째 위도 경도  
21401.21251958583 ← 거리  
: 컴퓨터 연구소 - 수원역
```



구글 지도

# Lab 2-0. Cities

- 파일에서 도시 정보 읽기
  - 도시 정보 모아둔 파일 cities.txt
  - 도시의 이름, 도시의 나라\t도시의 위도\t도시의 경도
  - 파일을 읽어서 DB 구축하기
  - Word Count 와 달리 dictionary 사용 {key:=name : val:=loc}

```
def get_cities()
```

# Lab 2-0. Cities

## ■ 구축한 DB에서 필요한 정보 찾기

- `get_cities()` 함수를 통해 .txt에 있는 정보를 프로그램으로 가져옴
- 내가 원하는 정보를 찾아내는 함수 만들기
- 도시의 이름을 넣어서
  - 해당 도시가 있는 경우 위치 좌표 반환 (위도, 경도)
  - 없는 경우 `None` 반환

```
def get_city_by_name(cities, name)
```

get\_cities() 함수에서 반환 받은 dict {name, loc}  
찾으려는 도시의 이름 str

# Lab 2-0. Cities

- 사용자의 요구에 따라 거리 알려주기

- 도시 정보 모아둔 파일 `cities.txt` 에 있는 도시 이름 두 개를 입력 받아 (순서에 상관 없이) 거리를 출력 (쉼표 “,” 로 구분)
- 이전에 구현한 함수들을 활용
- 단, `get_cities()` 함수는 전체 코드에서 단 한 번만 호출
- 22 page 의 입출력 예시 참고 :종료 조건, 예외 처리

```
def city_dist()
```

# Lab 2-0. Cities

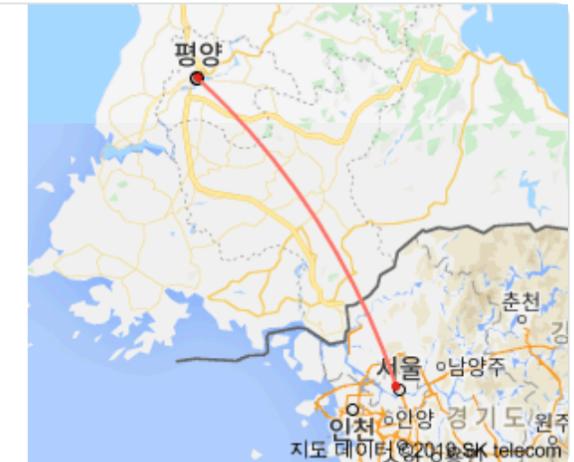
## ■ 서울에서 평양 거리

```
$ python cities.py  
Seoul, Pyeongyang ↵ ← ';'로 두 도시 구분  
239024.51825296393 (white space 주의)  
a, Seoul ↵  
There are no 'a'  
Paris, so far ↵  
There are no 'so far'  
EXITprogram ↵  
exit
```

왜 오차가 발생할까요?

195 km

서울특별시에서 평양까지의 거리



# Dict vs. Lists

# Lab 2+. Dict vs. Lists

- Dictionary 의 key 와 value

- Dictionary 의 key 는 str 뿐만 아니라 int, float, tuple 도 가능.  
단, list & dict는 불가능
- Dictionary 의 value 는 list & dict도 가능

```
>>> d = dict()↵
>>> d['my_key'] = 'my_value'↵
>>> d[3] = 3.21↵
>>> d[10.16] = [21, 'to', 22]↵
>>> d[(‘Lat’,1.25,(6,-2.0))] = ‘tuple’↵
>>> d[[1,2,3]] = ‘error’↵
TypeError: unhashable type: 'list'
```

# Lab 2+. Dict vs. Lists

- Dictionary 의 key 와 value

- 즉, dict 의 key 는 immutable 함 (해야 함.)
- 따라서 mutable 한 dict 꼴의 자료형이 필요할 경우,  
Word Count 에서 했듯, 두개의 list 를 대응되게 생성할 수 있음.  
(단, 느림 ← 왜?)

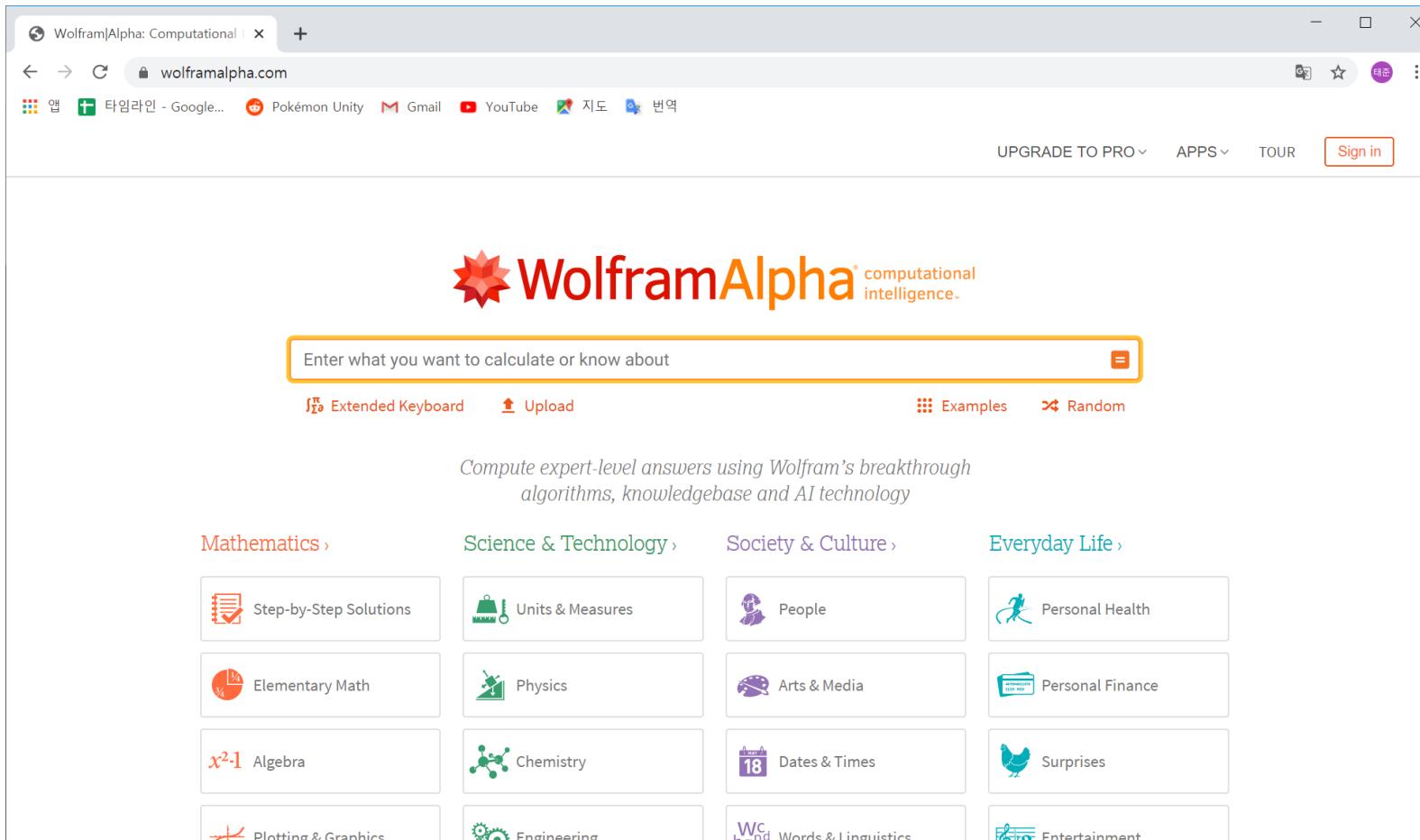
```
class mydict:  
    def __init__(self):  
        self.keys = list()  
        self.vals = list()  
  
    def get(self, key, default):  
    def sort(self):
```

# Wolfram-Alpha

# Lab 2-1.Wolfram-Beta

## ■ Wolfram-Alpha

- <https://www.wolframalpha.com/>



# Lab 2-1.Wolfram-Beta

## ■ Wolfram-Alpha

- <https://www.wolframalpha.com/>

The screenshot shows a web browser window for WolframAlpha. The address bar contains the URL <https://www.wolframalpha.com/input/?i=d+x^2+2x+1%2Fdx>. The main content area displays the result for the derivative of  $x^2 + 2x + 1$ , which is  $2x + 2$ . Below the result is a plot of the function  $y = x^2 + 2x + 1$  for  $x$  from -1.5 to 1.5, showing a parabola opening upwards with its vertex at (-1, 0). A promotional banner at the top right encourages users to "Relax. You have the answers." and offers an "Unlock Step-by-Step" option. The WolframAlpha logo is prominently displayed at the bottom.

# Wolfram-Beta

# Lab 2-I.Wolfram-Beta

- 최종적으로 만들고자 하는 그림
  - Alpha 는 아니어도, Beta 정도는 할 수 있다!

```
$ cat output.txt
D, x^2 + 2x + 1
I, 2x + 2, 1
C, x^3 + x^2 + x + 1, -1
$ python wolfram_beta.py
$ cat output.txt
2x + 2
x^2 + 2x + 1
0
```

# Lab 2-I.Wolfram-Beta

## ■ 최종적으로 만들고자 하는 그림

- 입력 파일에 있는 요청을 모두 처리하여 출력 파일 만들기
- 입력 요청의 종류 : 3가지

input.txt

```
D, f(x)  
I, f(x), c  
C, f(x), a
```

- 미분(Differential), 적분(Integral), 계산(Computation)

output.txt

```
f'(x)  
F(x) + c  
f(a)
```

# Lab 2-I. Wolfram-Beta

## ■ 구체적인 제약조건

### • 다항함수 문자열

- 다항함수 문자열은 최소 1개 이상의 항으로 구성되어 있어야 한다.
- 각각의 항은 덧셈('+') 기호로 구분되어 있으며, 덧셈 기호 사이에는 한 칸의 공백이 있다.
  - '[first\_term] + [second\_term] + ... + [last\_term]'  
– 항은 (정수) 계수와 (음이 아닌 정수) 차수로 표현할 수 있다.
    - 차수가 0인 경우, 이 항을 **상수항**이라 하며, 정수 계수를 있는 그대로 출력한다.(예: '-1', '0', '1')
    - 차수가 1 이상이면, 정수 계수 뒤에 문자 x를 붙인다.  
추가로, 차수가 2 이상이면, 문자 x 뒤에 기호 ^와 차수를 붙인다.(예: '-5x^2', '0x', '3x^4')
    - 차수가 1 이상이고 계수의 절댓값이 1인 경우, 숫자 1을 생략한다.(예: '-x', 'x^2', '-x^3')

# Lab 2-I. Wolfram-Beta

## ■ ‘D, f(x)’ 구체적인 제약조건

- 입력 받은 **다항함수 문자열**을 (입력 파일로 부터)  
‘미분한 결과’를 **다항함수 문자열**로 출력하기 (출력 파일로)
- **파일을 line 별로 해결**
  - Input 파일에서 ‘D’로 시작하는 줄은 미분을 요청함
  - 같은 식으로 ‘I’로 시작하는 줄은 적분을 요청,  
‘C’로 시작하는 줄은 계산을 요청
  - Output 파일의 같은 번째 줄에 해당 요청의 결과를 기록

# Lab 2-I. Wolfram-Beta

## ■ ‘D, f(x)’ 구체적인 제약조건

- 미분함수
  - 어느 임의의 다항 함수의 미분 함수는 다항 함수이다.
  - → **미분 함수 역시 여러 다항함수 문자열**로 표현 가능하다.
- 예 : `equation = 'x^2 + 2x + 1'`
  - `'2x + 2 + 0'`
  - `'2x + 2'`
  - `'2 + 2x'`
  - `'1 + x + 1 + x + 0x^2 + 0x^3'`
- 채점 코드는 항의 순서, 계수가 0인 항, 동류항 묶기 등등을 신경 쓰지 않는다!
- 신경 쓸만한 부분은, 다항함수 문자열은 **최소 1개의 항**이 필요하다는 점
  - 빈 문자열 `''`을 출력하면 안 된다. 최소한 `'0'`은 출력해야 함

# Lab 2-I.Wolfram-Beta

- **def print\_term(degree, factor):**

- *degree* : 항의 차수 (0 이상의 정수)
  - 0인 경우?
  - 1인 경우?
- *factor* : 항의 계수 (정수)
  - 음수인 경우?
  - 절댓값이 1인 경우?

```
assert print_term(2, 0) == "0x^2"
assert print_term(1, -1) == "-x"
assert print_term(0, 5) == "5"
```

# Lab 2-I.Wolfram-Beta

- `def print_equation(terms):`
  - *terms* : dict
    - dictionary {degree (int) : factor (int)}의 형태
    - key := degree (차수), value := factor (계수)
  - Expected output
    - str 함수를 문자열로 표현  $\leftarrow ' + '$  문자를 기준으로 합침 : `string.join()`?

```
assert print_equation(  
    {2:0, 1:-1, 0:5}  
) == "0x^2 + -x + 5"
```

# Lab 2-I.Wolfram-Beta

- `def parse_term(term_str):`

- `print_term()`의 역함수 꼴
- `term_str` : 항을 문자열로 표현
- Expected outputs
  - tuple := (degree : int, factor : int)
  - (차수, 계수) 순서쌍

```
assert parse_term("0x^2") == (2, 0)
assert parse_term("-x") == (1, -1)
assert parse_term("5") == (0, 5)
```

# Lab 2-I.Wolfram-Beta

- `def parse_equation(equation):`
  - *equation* : str
    - '+' + '-' 문자를 기준으로 쪼개기 : `string.split()`?  
White Space 주의
  - Expected output
    - dictionary {degree (int) : factor (int)}의 형태
    - key := degree (차수), value := factor (계수)

```
assert parse_equation("0x^2 + -x + 5") == \
{2:0, 1:-1, 0:5}
```

# Lab 2-I.Wolfram-Beta

## ■ `def d_dx_as_terms(terms):`

- *terms* : dict
  - dictionary {degree (int) : factor (int)}의 형태
  - {1항 차수 : 1항 계수, 2항 차수 : 2항 계수, ..., N항 차수 : N항 계수}
- Expected output
  - 형식은 *terms*와 동일
  - 가능한 답은 여러가지가 될 수 있음!
  - 신경 쓸 부분 (**다항함수 조건**)
    - 출력된 계수는 정수인가
    - 출력된 차수는 0 이상의 정수인가
    - 항의 개수는 1 이상인가
    - 미분 법칙에 잘 맞는가 :  $(\frac{d}{dx} ax^n = (an)x^{n-1}, \frac{d}{dx} a = 0 \text{ and } \frac{d}{dx} (f + g) = \frac{d}{dx} f + \frac{d}{dx} g)$

# Lab 2-I.Wolfram-Beta

- **def d\_dx(equation):**

- *equation* : str
- Expected output : str
- 지금껏 구현했던 함수들을 총동원하기
- 3줄로 간결하게 표현할 수 있음

```
assert d_dx("0x^2 + -x + 5") == "2x + -1"
```

# Lab 2-I. Wolfram-Beta

- ‘I, f(x), c’ 구체적인 제약조건
  - 입력 받은 **다항함수 문자열**을 (입력 파일로 부터)  
‘적분한 결과’를 **다항함수 문자열**로 출력하기 (출력 파일로)
  - ‘c’
    - 적분 상수
    - 정수 (int)

# Lab 2-I. Wolfram-Beta

## ■ ‘I, f(x), c’ 구체적인 제약조건

- 적분함수
  - 어느 임의의 다항 함수의 적분 함수는 다항 함수이다.
  - → **적분 함수 역시 여러 다항함수 문자열**로 표현 가능하다.
- 예 : `equation = '2x + 2'`
  - `'x^2 + 2x + c'`
  - `'x^2 + x + x + c'`
  - `'2x + x^2 + c'`
  - `'c + x + x^2 + x + 0x^3'`
- 채점 코드는 항의 순서, 계수가 0인 항, 동류항 묶기 등등을 신경 쓰지 않는다!
- 신경 쓸만한 부분은, 다항함수 문자열은 **최소 1개의 항**이 필요하다는 점
  - 빈 문자열 ''을 출력하면 안 된다. 최소한 '0'은 출력해야 함

# Lab 2-I.Wolfram-Beta

- `def integral_as_terms(terms, constant):`
  - *terms* : dict
    - `d_dx_as_terms` 의 *terms* 와 동일
  - *terms* : constant
    - ‘int’ type, 적분 상수
  - Expected output
    - 형식은 *terms*와 동일
    - 가능한 답은 여러가지가 될 수 있음!
    - 신경 쓸 부분 (**다항함수 조건**)
      - 출력된 계수는 정수인가
      - 출력된 차수는 0 이상의 정수인가
      - 항의 개수는 1 이상인가
      - 적분 법칙에 잘 맞는가 :  $(ax^n = \int(an)x^{n-1}dx, a = \int 0, \text{ and } \int(f + g) = \int f + \int g)$

# Lab 2-I.Wolfram-Beta

- **def integral(equation, constant):**

- *equation* : str
- *constant* : int
- Expected output : str
- 지금껏 구현했던 함수들을 총동원하기
- 3줄로 간결하게 표현할 수 있음

```
assert integral("2x + -1", 5) == "x^2 + -x + 5"
```

# Lab 2-I. Wolfram-Beta

## ■ ‘ $C, f(x), a$ ’ 구체적인 제약조건

- 입력 받은 **다항함수 문자열**을 (입력 파일로 부터)  
함수로 전환해 ‘정수  $a$  를 대입한 결과’를 **정수**로 출력하기 (출력 파일로)
- ‘ $a$ ’
  - 0이 아닌 정수

# Lab 2-I.Wolfram-Beta

- `def compute_as_terms(terms, x):`
  - *terms* : dict
    - `d_dx_as_terms` 의 *terms* 와 동일
  - *terms* : x
    - ‘int’ type, 대입 상수 (0이 아님)
  - Expected output
    - ‘int’ type
    - 신경 쓸 부분
      - 계산된 결과는 정수인가?

# Lab 2-I.Wolfram-Beta

- **def compute(equation, x):**

- *equation* : str
  - *x*: int ( $\neq 0$ )
  - Expected output : string
- 
- 지금껏 구현했던 함수들을 총동원하기
  - 구현한 함수들로 간결하게 표현할 수 있음

```
assert compute("2x + -1", 5) == "9"
```

# Lab 2-I.Wolfram-Beta

- `def solve_query(line):`

- `line : str`
- `Expected output : str`
- 현재, 처리하고 있는 줄 ‘`line`’이 어떤 요청인지 판단해서
- 그에 대응하는 응답을 반환함
- 이미 구현한 함수를 이용하면 간결하게 표현할 수 있음

```
assert solve_query("D,0x^2 + -x + 5") == "2x + -1"
assert solve_query("I,2x + -1,5") == "x^2 + -x + 5"
assert solve_query("C,2x + -1,5") == 9
```

# Lab 2-I.Wolfram-Beta

## ■ 파일 쓰기

```
f = open("./test.txt", "w")
f.write("Hello\n")
f.write("My")
f.write("World!")
f.close()
```

### - 실행 결과

**./test.txt**

```
Hello
MyWorld!
```

# Lab 2-I.Wolfram-Beta

- `def solve(input_path, output_path):`
  - *path* : str
  - Expected output : None
- ‘`input_path`’에는 1 줄에 1 개의 요청이 적힌 파일이 있음
- 함수 호출 후 ‘`output_path`’에 파일 만들
- 만들어진 파일의 각 줄에는 입력 받은 파일의 요청에 대한 응답이 있음
- 즉, 입력 파일 10번째 줄이 ‘ $D, 0x^2 + -x + 5$ ’라면,  
만들어진 출력파일 10번째 줄은 ‘ $2x + -1$ ’
- `word_count, cities` 실습처럼 파일에서 정보를 읽고 사용하는 모듈

# Lab 2-I.Wolfram-Beta

## ■ Test-Cases

- 총 100 개의 Test-Case 준비 됨  
설명 드린 대로 다행 함수만 존재, 잘못된 입력 없음 (ex ‘C, x^-1, 0’)
- input\_sample.txt
- answer\_sample.txt

# Wolfram-Beta ØØ

# Lab 2-2. Wolfram-Beta ØØ

- Wolfram-Beta 를 Object Oriented 로 변환
  - 인터페이스 영역 (WolframBeta) 수식 영역 (Terms) 나누기  
class 내장 함수의 첫 번째 parameter 는 self  
class 내장 함수에서 다른 내장 함수나 내장 변수를 호출 할 때는 self.func  
(str.strip().split() 과 같음)
  - 다항 함수 뿐만 아니라 cos, sin, exp 세 개의 함수에 대해서도 지원  
→ math 라이브러리를 사용하여 C (compute) 쿼리 수행
  - 정수 뿐만 아니라 실수 계수, 차수 지원

# Lab 2-2. Wolfram-Beta ØØ

## ■ Object Oriented : Class

- Terms
  - `__init__`, `__str__`, `__eq__`
  - `self.terms_dict`
  - `self.equation`
  - `print_term(degree, factor)` **@static**
  - `print_equeation(self, equation)`
  - `parse_term(term_str)` **@static**
  - `parse_equeation(self, equation)`
  - `d_dx_as_terms(self)`
  - `integral_as_terms(self, constant)`
  - `compute_as_terms(self, x)`
- WolframBeta
  - `__init__`
  - `self.input_path`
  - `self.output_path`
  - `d_dx(equation)` **@static**
  - `integral(equation, constant)` **@static**
  - `compute(equation, x)` **@static**
  - `solve_query(self, line)`
  - `solve(self)`

# Lab 2-2. Wolfram-Beta ØØ

## ■ 추가 기능 지원 ( optional )

- math 라이브러리 사용
- $\cos(x)$ ,  $\sin(x)$ ,  $\exp(x)$ 
  - Degree 에 차수(int)가 아닌 함수 이름(str) 이 들어감.
  - type(x) is str -> 문자열인지 아닌지 판별
- 실수형 계수 (int 아닌 float)
- 실수형 차수 (int 아닌 float)
  - 음수 차수 (단 -1 제외)
  - N.m 차수 (\*\* 사용)
- I 또는 C 요청에서에서 적분 상수 c, 입력 값 a에 실수 (float)

# Lab 2-2. Wolfram-Beta ØØ

## ■ Test-Cases

- 총 1120 개의 Test-Case 준비 됨  
앞선 Test-Cases 1000개 + 추가 기능에 대한 Test 120개
- 상대 오차, 절대 오차  $10^{-6}$  까지 허용  
뼈대 코드의 Terms의 `__eq__` 참고 (수정 X)
- `sys.snu.ac.kr` 을 통해 제출  
1120 점 만점 틀린 case 중 일부만 보여줌.  
뼈대 코드 내 test 는 `log.txt` 에 모두 보여줌.

실습 서버

# sys.snu.ac.kr : 실습 서버

## ■ Login & Register

The image displays two adjacent browser windows side-by-side, both showing the "SNU: Systems Software & Architecture Lab." website.

**Left Window (Login Page):**

- Header:** SNU: Systems Software & Architecture Lab.
- Title:** Login
- Fields:** Email (text input), Password (text input).
- Buttons:** Sign in (blue button), Join (gray button).
- Text at bottom:** We recommend the Google Chrome web browser.  
© SNU Systems Software & Architecture Laboratory

**Right Window (Join Page):**

- Header:** sys.snu.ac.kr
- Title:** Join
- Fields:** Email (text input), Password (text input), Password Confirm (text input), Name (Real Name) (text input), NickName (text input), Student No. (text input), Mobile (text input - 010 - 0000 - 0000).
- Buttons:** Register (blue button), Cancel (gray button).
- Text at bottom:** We recommend the Google Chrome web browser.  
© SNU Systems Software & Architecture Laboratory

# sys.snu.ac.kr : 실습 서버

## ■ Main Page

The screenshot shows a web browser window for the URL `sys.snu.ac.kr`. The title bar reads "SNU: Systems Software & Architecture Lab.". The main content area displays the "Python for Data Analytics" class page. On the left, a sidebar menu lists "CLASS LIST" (with "[PyDA] Python for Data Analytics" selected), "CLASS HOME" (with "Overview" selected), and "SYSTEM" sections, along with links for "Logout". The main content area features a large heading "Python for Data Analytics" and a blue button "Goto Homepage ». Below this, a light blue box contains the text "[2020/01/08]" and "Word Count with List has been posted.", followed by the date "Date : 2019.01.07 13:30:00 ~ 2019.01.08 18:00:00". At the bottom of the page, there is a note about recommended browsers and copyright information.

SNU: Systems Software & Architecture Lab.

CLASS LIST  
[PyDA] Python for Data Analytics

CLASS HOME  
Overview

Projects  
Rank  
Latest Runs  
QnA  
Course Homepage

SYSTEM  
Test Queue Status  
Member Info  
Logout

**Python for Data Analytics**

Goto Homepage »

[2020/01/08]

Word Count with List has been posted.

Date : 2019.01.07 13:30:00 ~ 2019.01.08 18:00:00

We recommend the Google Chrome web browser.

© SNU Systems Software & Architecture Laboratory

# sys.snu.ac.kr : 실습 서버

## ■ Project Page

SNU: Systems Software & Architecture Lab.

CLASS LIST  
[PyDA] Python for Data Analytics

CLASS HOME  
Overview  
Projects  
Rank  
Latest Runs  
QnA  
Course Homepage

SYSTEM  
Test Queue Status  
Member Info  
Logout

We recommend the Google Chrome web browser.  
© SNU Systems Software & Architecture Laboratory

SNU: Systems Software & Architecture Lab.

CLASS LIST  
[PyDA] Python for Data Analytics

CLASS HOME  
Overview  
Projects  
Rank  
Latest Runs  
QnA  
Course Homepage

SYSTEM  
Test Queue Status  
Member Info  
Logout

PyDA : Projects Submit

Word Count with List

dict 없이 list 만으로 단어의 빈도수 세기  
Due date : 2019/01/08 18:00:00

[View details »](#) [Template Code »](#)

Source Code :  
Report :

#	Date	File	Compile	Execution	Final	Metric	Test	-
No submission								

We recommend the Google Chrome web browser.  
© SNU Systems Software & Architecture Laboratory